1/3/1 (Item 1 from file: 351)
DIALOG(R)File 351:Derwent WPI

0013276065 - Drawing available
WPI ACC NO: 2003-362161/200334
XRPX Acc No: N2003-289206
Multi-thread execution method for parallel processing system, involves determining forkability of slave thread and executing fork instructions of slave thread or subsequent instruction of master thread
Patent Assignee: NEC CORP (NIDE)
Inventor: MATSUSHITA S; OHSAWA T; OSAWA H
Patent Family (5 patents, 3 countries)

Patent Number	Kind	Date	Application Number	Kind	Date	Update	
US 20030014471	A1	20030116	US 2002133409	A	20020429	200334	B
JP 2003029984	A	20030131	JP 2001212246	A	20010712	200334	E
GB 2381610	A	20030507	GB 200216272	A	20020712	200338	E
JP 3702813	B2	20051005	JP 2001212246	A	20010712	200565	E
GB 2381610	B	20060222	GB 200216272	A	20020712	200615	E

Priority Applications (no., kind, date): JP 2001212246 A 20010712

Patent Details

Number	Kind	Lan	Pg	Dwg	Filing Notes
US 20030014471	A1	EN	40	24	
JP 2003029984	A	JA	21		
JP 3702813	B2	JA	28		Previously issued patent JP 2003029984

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2003-029984
(43)Date of publication of application : 31.01.2003

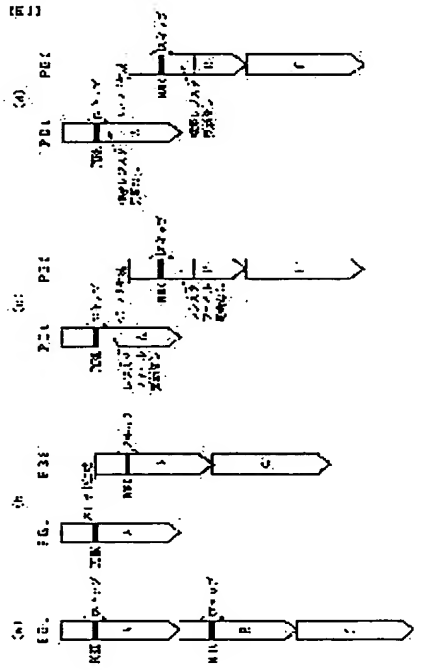
(51)Int.Cl. G06F 9/46
G06F 15/16
G06F 15/177

(21)Application number : 2001-212246
(22)Date of filing : 12.07.2001

(71)Applicant : NEC CORP
(72)Inventor : OSAWA HIROSHI
MATSUSHITA SATOSHI

(54) MULTITHREAD EXECUTION METHOD AND PARALLEL PROCESSOR SYSTEM

(57)Abstract:



PROBLEM TO BE SOLVED: To provide a multithread execution method executing the processing of a program without failure and without the interruption of the processing even in the case that a free processor capable of generating a slave thread is not present at the point in time of the fork instruction of a master thread while preventing the increase of the amount of hardware due to the provision of many register files and the increase of overheads at the time of the process changeover of an OS.

SOLUTION: At the time of dividing a single program into a plurality of threads A-C and parallelly executing them in a plurality of processors PE1 and PE2, the forking possibility of the slave thread B to the other processor PE2 by the fork instruction in the master thread A executed in the processor PE1 is judged and the slave thread B is forked to the processor PE2 when forking is possible. When forking is impossible, the fork instruction is invalidated, succeeding instructions after the fork instruction are continuously executed in the processor PE1 and then the instruction group of the slave thread B is executed in the processor PE1.

LEGAL STATUS

[Date of request for examination] 17.01.2002
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number] 3702813
[Date of registration] 29.07.2005
[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2003－29984
（P2003－29984A）
(43)公開日　平成15年１月31日(2003.1.31)

(51)Int.Cl.⁷	識別記号	ＦＩ	テーマコード（参考）
Ｇ０６Ｆ　9/46	３６０	Ｇ０６Ｆ　9/46	３６０Ｂ　５Ｂ０４５
15/16	６１０	15/16	６１０Ｚ　５Ｂ０９８
15/177	６７４	15/177	６７４Ａ
	６８１		６８１Ｂ

審査請求　有　　請求項の数22　ＯＬ　（全 21 頁）

(21)出願番号　特願2001－212246(P2001－212246)

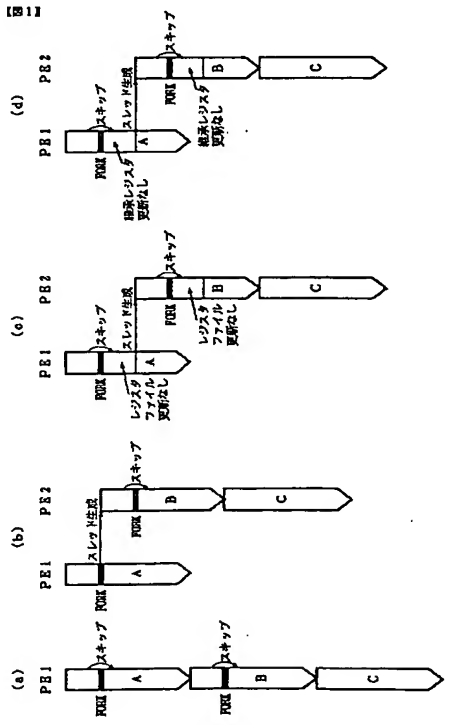
(22)出願日　平成13年７月12日(2001.7.12)

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号
(72)発明者　大澤　拓
東京都港区芝五丁目７番１号　日本電気株式会社内
(72)発明者　松下　智
東京都港区芝五丁目７番１号　日本電気株式会社内
(74)代理人　100088959
弁理士　境　廣巳

Ｆターム(参考)　5B045 GG11
5B098 AA10 GA05 GC14

(54)【発明の名称】　マルチスレッド実行方法及び並列プロセッサシステム

(57)【要約】
【課題】　多数のレジスタファイルを持つことによるハードウェア量の増加、ＯＳのプロセス切り替え時におけるオーバヘッドの増大を防止しつつ、親スレッドのフォーク命令時点で子スレッドを生成できる空きのプロセッサが存在しない場合でも処理の中断無しにプログラムの処理を支障なく遂行できるマルチスレッド実行方法を提供する。
【解決手段】　単一のプログラムを複数のスレッドＡ～Ｃに分割し、複数のプロセッサＰＥ１、ＰＥ２で並列に実行する際、プロセッサＰＥ１で実行している親スレッドＡ中のフォーク命令による他プロセッサＰＥ２への子スレッドＢのフォーク可能性を判定し、フォーク可能ならば子スレッドＢをプロセッサＰＥ２にフォークし、フォーク不可能ならば、フォーク命令を無効化してプロセッサＰＥ１でフォーク命令以降の後続命令を引き続き実行した後、子スレッドＢの命令群をプロセッサＰＥ１で実行する。



【特許請求の範囲】

【請求項１】　単一のプログラムを複数のスレッドに分割し複数のプロセッサで並列に実行するマルチスレッド実行方法において、

親スレッド中のフォーク命令による他プロセッサへの子スレッドのフォーク可能性を判定し、フォーク可能ならば前記子スレッドをフォークし、フォーク不可能ならば前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行した後に前記子スレッドの命令群を実行することを特徴とするマルチスレッド実行方法。

【請求項２】　前記フォーク可能性の判定を前記フォーク命令の時点でのみ実施する請求項１記載のマルチスレッド実行方法。

【請求項３】　前記フォーク可能性の判定は、前記フォーク命令の時点で前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行う請求項２記載のマルチスレッド実行方法。

【請求項４】　前記フォーク可能性の判定を前記フォーク命令の時点及びその時点でフォーク不可能と判定した場合には前記フォーク命令以降の時点でも実施する請求項１記載のマルチスレッド実行方法。

【請求項５】　前記フォーク命令の時点での前記フォーク可能性の判定は、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行い、前記フォーク命令以降の時点での前記フォーク可能性の判定は、前記親スレッドのレジスタファイルが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって行う請求項４記載のマルチスレッド実行方法。

【請求項６】　前記フォーク命令の時点での前記フォーク可能性の判定は、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行い、前記フォーク命令以降の時点での前記フォーク可能性の判定は、前記親スレッドのレジスタファイル中のレジスタのうち前記子スレッドに継承すべきレジスタが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって行う請求項４記載のマルチスレッド実行方法。

【請求項７】　前記子スレッドのフォークが行われたときに前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、前記フォークドビットがセットされた前記プロセッサで実行中のスレッドのターム命令は有効化し、前記フォークドビットがセットされない前記プロセッサで実行中のスレッドのターム命令は無効化する請求項１乃至６の何れか１項に記載のマルチスレッド実行方法。

【請求項８】　前記親スレッドのフォーク命令の時点でフォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存し、前記子スレッドのフォークが行われたときに前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了する請求項１乃至６の何れか１項に記載のマルチスレッド実行方法。

【請求項９】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）親スレッド中のフォーク命令の時点で前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、（ｂ）フォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令を無効化し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、（ｃ）前記フォークドビットがセットされた前記プロセッサで実行中のスレッドのターム命令は有効化し、前記フォークドビットがセットされない前記プロセッサで実行中のスレッドのターム命令は無効化するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１０】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）親スレッド中のフォーク命令の時点で、フォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存すると共に、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、（ｂ）フォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行

する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令を無効化し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、（ｃ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１１】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）親スレッド中のフォーク命令の時点で、フォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存すると共に、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、（ｂ）ステップａの判定がフォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令によるフォークを保留し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、（ｃ）ステップｂでフォークを保留した場合、前記フォーク命令以降の後続命令の実行と並行して、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記親スレッドのレジスタファイルの更新前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって判定し、フォーク可能と判定された時点で、前記レジスタに保存されたフォーク先アドレスに従って子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットするステップ、
（ｄ）前記フォークドビットがセットされた前記プロセッサで実行中のスレッドのターム命令は有効化し、前記フォークドビットがセットされない前記プロセッサで実行中のスレッドのターム命令は無効化するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１２】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）親スレッド中のフォーク命令の時点で、フォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存すると共に、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、（ｂ）ステップａの判定がフォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令によるフォークを保留し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、（ｃ）ステップａでフォーク不可能と判定された場合、前記フォーク命令以降の後続命令の実行と並行して、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記親スレッドのレジスタファイルの更新前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって判定し、フォーク可能となった時点で、前記レジスタに保存されたフォーク先アドレスに従って子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットするステップ、
（ｄ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１３】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）親スレッド中のフォーク命令の時点で、フォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存すると共に、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、（ｂ）ステップａの判定

がフォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令によるフォークを保留し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、（ｃ）ステップａでフォーク不可能と判定された場合、前記フォーク命令以降の後続命令の実行と並行して、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記親スレッドのレジスタファイル中のレジスタのうち前記子スレッドに継承すべきレジスタが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって判定し、フォーク可能と判定された時点で、前記レジスタに保存されたフォーク先アドレスに従って子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットするステップ、（ｄ）前記フォークドビットがセットされた前記プロセッサで実行中のスレッドのターム命令は有効化し、前記フォークドビットがセットされない前記プロセッサで実行中のスレッドのターム命令は無効化するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１４】　各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドで必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、（ａ）親スレッド中のフォーク命令の時点で、フォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存すると共に、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、（ｂ）ステップａの判定がフォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令によるフォークを保留し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、（ｃ）ステップａでフォーク不可能と判定された場合、前記フォーク命令以降の後続命令の実行と並行して、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記親スレッドのレジスタファイル中のレジスタのうち前記子スレッドに継承すべきレジスタが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって判定し、フォーク可能と判定された時点で、前記レジスタに保存されたフォーク先アドレスに従って子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットするステップ、（ｄ）前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項１５】　単一のプログラムを複数のスレッドに分割し複数のプロセッサで並列に実行する並列プロセッサシステムにおいて、親スレッド中のフォーク命令による他プロセッサへの子スレッドのフォーク可能性を判定する手段と、フォーク可能ならば前記子スレッドをフォークし、フォーク不可能ならば前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行した後に前記子スレッドの命令群を実行する手段とを備えたことを特徴とする並列プロセッサシステム。

【請求項１６】　前記フォーク可能性の判定を前記フォーク命令の時点でのみ実施する構成を有する請求項１５記載の並列プロセッサシステム。

【請求項１７】　前記フォーク可能性の判定は、前記フォーク命令の時点で前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行う請求項１６記載の並列プロセッサシステム。

【請求項１８】　前記フォーク可能性の判定を前記フォーク命令の時点及びその時点でフォーク不可能と判定した場合には前記フォーク命令以降の時点でも実施する構成を有する請求項１５記載の並列プロセッサシステム。

【請求項１９】　前記フォーク命令の時点での前記フォーク可能性の判定は、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行い、前記フォーク命令以降の時点での前記フォーク可能性の判定は、前記親スレッドのレジスタファイルが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって行う請求項１８記載の並列プロセッサシステム。

【請求項２０】　前記フォーク命令の時点での前記フォーク可能性の判定は、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行い、前記フォーク命令以降の時点での前記フォーク可能性の判定は、前記親スレッドのレジスタファイル中のレジスタのうち前記子スレッドに継承すべきレジスタが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって行う請求項１８記載の並列プロセッサシステム。

【請求項２１】　前記子スレッドのフォークが行われたときに前記親スレッドを実行する前記プロセッサに設け

たフォークドビットをセットする手段と、前記フォークドビットがセットされた前記プロセッサで実行中のスレッドのターム命令は有効化し、前記フォークドビットがセットされない前記プロセッサで実行中のスレッドのターム命令は無効化する手段とを備えた請求項１５乃至２０の何れか１項に記載の並列プロセッサシステム。

【請求項２２】　前記親スレッドのフォーク命令の時点でフォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存する手段と、前記子スレッドのフォークが行われたときに前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットする手段とを備え、前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了する構成を有する請求項１５乃至２０の何れか１項に記載の並列プロセッサシステム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は並列プロセッサシステムにおけるプログラム並列実行方法に関し、より具体的には単一のプログラムを複数のスレッドに分割して複数のプロセッサにより並列に実行するマルチスレッド実行方法及び並列プロセッサシステムに関する。

【０００２】

【従来の技術】単一のプログラムを並列プロセッサシステムで並列に処理する手法として、プログラムをスレッドと呼ぶ命令流に分割して複数のプロセッサで並列に実行するマルチスレッド実行方法があり、この方法を記載した文献として、特開平１０－２７１０８号公報（以下、文献１と称す）、「Ｏｎ　Ｃｈｉｐ　Ｍｕｌｔｉｐｒｏｃｅｓｓｏｒ指向　制御並列アーキテクチャＭＵＳＣＡＴの提案」（並列処理シンポジュウムＪＳＰＰ９７論文集、情報処理学会、ｐｐ．２２９－２３６、Ｍａｙ　１９９７）（以下、文献２と称す）、特開平１０－７８８８０号公報（以下、文献３と称す）等がある。以下、これらの従来文献に記載されたマルチスレッド実行方法について説明する。

【０００３】一般にマルチスレッド実行方法において、他のプロセッサ上に新たなスレッドを生成することを、スレッドをフォーク（ｆｏｒｋ）すると言い、フォーク動作を行った側のスレッドを親スレッド、生成された新しいスレッドを子スレッド、スレッドをフォークする箇所をフォーク点、子スレッドの先頭箇所をフォーク先アドレスまたは子スレッドの開始点と呼ぶ。文献１～３では、スレッドのフォークを指示するためにフォーク点にフォーク命令が挿入される。フォーク命令にはフォーク先アドレスが指定され、フォーク命令の実行によりそのフォーク先アドレスから始まる子スレッドが他プロセッサ上に生成され、子スレッドの実行が開始される。また、スレッドの処理を終了させるターム（ｔｅｒｍ）命令と呼ばれる命令が用意されており、各プロセッサはターム命令を実行することによりスレッドの処理を終了する。

【０００４】図２２に従来のマルチスレッド実行方法の処理の概要を示す。同図（ａ）は３つのスレッドＡ、Ｂ、Ｃに分割された単一のプログラムを示す。このプログラムを単一のプロセッサで処理する場合、同図（ｂ）に示すように１つのプロセッサＰＥがスレッドＡ、Ｂ、Ｃを順番に処理していく。これに対して文献１～３のマルチスレッド実行方法では、同図（ｃ）に示すように、１つのプロセッサＰＥ１にスレッドＡを実行させ、プロセッサＰＥ１でスレッドＡを実行している最中に、スレッドＡに埋め込まれたフォーク命令によってスレッドＢを他のプロセッサＰＥ２に生成し、プロセッサＰＥ２においてスレッドＢを実行させる。また、プロセッサＰＥ２はスレッドＢに埋め込まれたフォーク命令によってスレッドＣをプロセッサＰＥ３に生成する。プロセッサＰＥ１、ＰＥ２はそれぞれスレッドＢ、Ｃの開始点の直前に埋め込まれたターム命令によってスレッドの処理を終了し、プロセッサＰＥ３はスレッドＣの最後の命令を実行すると、その次の命令（一般にはシステムコール命令）を実行する。このように複数のプロセッサでスレッドを同時に並行して実行することにより、逐次処理に比べて性能の向上が図られる。

【０００５】従来の他のマルチスレッド実行方法として、図２２（ｄ）に示すように、スレッドＡを実行しているプロセッサＰＥ１からフォークを複数回行うことにより、プロセッサＰＥ２にスレッドＢを、またプロセッサＰＥ３にスレッドＣをそれぞれ生成するマルチスレッド実行方法も存在する。この図２２（ｄ）のモデルに対して、図２２（ｃ）に示したようにスレッドはその生存中に高々１回に限って有効な子スレッドを生成することができるという制約を課したマルチスレッド実行方法をフォーク１回モデルと呼ぶ。フォーク１回モデルでは、スレッド管理の大幅な簡略化が可能となり、現実的なハードウェア規模でスレッド管理部のハードウェア化が実現できる。本発明はこのようなフォーク１回モデルを前提とする。

【０００６】ここで、フォーク命令時、子スレッドを生成できる空きのプロセッサが存在しない場合、従来は次の２通りの方法の何れかを採用している。

（１）親スレッドを実行しているプロセッサは、子スレッドを生成できる空きのプロセッサが生じるまで、フォーク命令の実行をウエイトする。

（２）親スレッドを実行しているプロセッサは、フォーク点におけるレジスタファイルの内容（フォーク先アドレス及びレジスタ内容）を裏面の物理レジスタに保存して親スレッドの後続処理を続行する。裏面の物理レジスタに保存されたレジスタファイルの内容は、子スレッド

を生成できる空きのプロセッサが生じた時点で参照され、子スレッドが生成される。

【０００７】親スレッドが子スレッドを生成し、子スレッドに所定の処理を行わせるには、親スレッドのフォーク点におけるレジスタファイル中のレジスタのうち少なくとも子スレッドで必要なレジスタの値を親スレッドから子スレッドに引き渡す必要がある。このスレッド間のデータ引き渡しコストを削減するために、文献２及び３では、スレッド生成時のレジスタ値継承機構をハードウェア的に備えている。これは、スレッド生成時に親スレッドのレジスタファイルの内容を子スレッドに全てコピーするものである。子スレッド生成後は、親スレッドと子スレッドのレジスタ値の変更は独立となり、レジスタを用いたスレッド間のデータの引き渡しは行われない。スレッド間のデータ引き渡しに関する他の従来技術としては、レジスタの値を命令によりレジスタ単位で個別に転送する機構を備えた並列プロセッサシステムも提案されている。

【０００８】その他、文献２に記載のＭＵＳＣＡＴでは、スレッド間の同期命令など、スレッドの並列動作を柔軟に制御するための専用命令が数多く用意されている。

【０００９】

【発明が解決しようとする課題】上述したように従来のマルチスレッド実行方法においては、子スレッドを生成できる空きのプロセッサが存在しない場合、空きのプロセッサが生じるまで親スレッドのフォーク命令の実行をウエイトしているため、場合によっては長時間待たされ、処理効率が極端に低下するという課題がある。

【００１０】この処理停止による処理効率の低下を改善するため、フォーク点におけるレジスタファイルの内容を裏面の物理レジスタに保存して親スレッドの処理の続行を可能にする方法では、各プロセッサ毎に表裏の少なくとも２面のレジスタファイルが必要になる。１つのレジスタファイルに例えば３２ビットのレジスタが３２個収納されているとすると、１プロセッサ当たり３２×３２ビットのメモリが必要になり、１チップにｎ個のプロセッサを集積化したオンチップ並列プロセッサでは無視できないハードウェア量の増加となる。また、単にハードウェア量の増加だけでなく、オペレーティングシステム（ＯＳ）によるプロセス切り替え時には裏面の物理レジスタも表の物理レジスタと一緒に退避、復元する必要があるため、プロセス切り替え時の処理量が増大し、プロセス切り替え時のオーバヘッドの増大による性能の低下を招く。

【００１１】また上述した従来のマルチスレッド実行方法においては、スレッドを終了させるためには必ずターム命令を子スレッドの開始点の直前に記述しておく必要がある。ターム命令は１スレッド当たり１個必要になるため、１つのスレッドに含まれる命令数が少ない細粒度スレッドほど、全命令数に占めるターム命令の割合が多くなる。ターム命令も他の命令と同様に命令メモリに格納されてフェッチの対象となるため、命令メモリのハードウェア量の増加、命令フェッチ数の増加による処理性能の低下が問題となる。

【００１２】本発明はこのような従来の問題点を解決したものであり、その目的は、多数のレジスタファイルを持つことによるハードウェア量の増加、ＯＳのプロセス切り替え時におけるオーバヘッドの増大を防止しつつ、親スレッドのフォーク命令時点で子スレッドを生成できる空きのプロセッサが存在しない場合でも処理の中断無しにプログラムの処理を支障なく遂行できる新規なマルチスレッド実行方法及び並列プロセッサシステムを提供することにある。

【００１３】また本発明の別の目的は、スレッドを終了させるためのターム命令を削減することにより命令メモリに必要な容量を削減し、また命令フェッチ数の削減による処理性能の向上を図ることにある。

【００１４】

【課題を解決するための手段】本発明は、単一のプログラムを複数のスレッドに分割し複数のプロセッサで並列に実行する際、親スレッド中のフォーク命令による他プロセッサへの子スレッドのフォーク可能性を判定し、フォーク可能ならば子スレッドをフォークし、フォーク不可能ならば親スレッドを実行しているプロセッサでフォーク命令以降の後続命令を引き続き実行した後に子スレッドの命令群を実行することを基本とする。

【００１５】より具体的には、第１の発明は、親スレッド中のフォーク命令による子スレッドのフォーク可能性の判定を、子スレッドの実行を開始できる他のプロセッサが存在するか否かによって、フォーク命令の時点でのみ実施し、可能ならばフォークし、その時点でフォーク不可能であれば当該子スレッドのフォークは断念して、子スレッドの命令群は親スレッドを実行しているプロセッサで実行する。

【００１６】また、第２の発明は、フォーク可能性の判定を子スレッドの実行を開始できる他のプロセッサが存在するか否かによってフォーク命令の時点で判定し、可能ならばフォークし、その時点でフォーク不可能であってもフォークを即断念せずに保留し、フォーク命令以降の後続命令の処理と並行して、親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他のプロセッサが生じたか否かによってフォーク可能性の判定を続け、フォーク可能となった時点でフォークし、最終的にフォーク不可能ならば当該子スレッドのフォークは断念して、子スレッドの命令群は親スレッドを実行しているプロセッサで実行する。

【００１７】また、第３の発明は、フォーク可能性の判定を子スレッドの実行を開始できる他のプロセッサが存在するか否かによってフォーク命令の時点で判定し、可

能ならばフォークし、その時点でフォーク不可能であってもフォークは即断念せずに保留し、フォーク命令以降の後続命令の処理と並行して、親スレッドのレジスタファイル中のレジスタのうち子スレッドに継承すべきレジスタが更新される前に子スレッドの実行を開始できる他のプロセッサが生じたか否かによってフォーク可能性の判定を続け、フォーク可能となった時点でフォークし、最終的にフォーク不可能ならば当該子スレッドのフォークは断念して、子スレッドの命令群は親スレッドを実行しているプロセッサで実行する。

【００１８】また、第４の発明は、子スレッドのフォークが行われたときに親スレッドを実行するプロセッサに設けたフォークドビットをセットし、フォークドビットがセットされたプロセッサで実行中のスレッドのターム命令は有効化し、フォークドビットがセットされないプロセッサで実行中のスレッドのターム命令は無効化する。

【００１９】また、第５の発明は、親スレッドのフォーク命令の時点でフォーク先アドレスを親スレッドを実行するプロセッサに設けたレジスタに保存し、子スレッドのフォークが行われたときに親スレッドを実行するプロセッサに設けたフォークドビットをセットし、フォークドビットがセットされており且つプログラムカウンタの値がレジスタに保存されたフォーク先アドレスと一致したプロセッサはスレッドの処理を終了する。

【００２０】

【作用】本発明にあっては、親スレッド中のフォーク命令による他プロセッサへの子スレッドのフォーク可能性を判定し、フォーク不可能ならば、親スレッドを実行しているプロセッサで後続命令を引き続き実行するため、処理が中断することがなく、また、親スレッドの後続命令を実行した後に子スレッドの命令群を実行するため、プログラムの処理を支障なく遂行でき、更に、子スレッドの処理を親スレッドと同じプロセッサで行うため、裏面のレジスタファイルにフォーク点のレジスタファイルの内容を退避しておく必要がなくなり、多数のレジスタファイルを持つことによるハードウェア量の増加、ＯＳのプロセス切り替え時におけるオーバヘッドの増大を防止できる。

【００２１】以下、図２２（ａ）に示した３つのスレッドＡ、Ｂ、Ｃに分割された単一のプログラムを例に本発明の作用を説明する。

【００２２】（１）第１の発明

図１（ａ）に示すように、１つのプロセッサＰＥ１にスレッドＡを実行させ、プロセッサＰＥ１でスレッドＡを実行しているときにフォーク命令に実行が差しかかると、子スレッドＢの実行を開始することができる他プロセッサが存在するか否かが判定される。図１（ａ）では、そのようなプロセッサが存在しないため、当該フォーク命令は無効化され、後続命令の処理が引き続き実行され、且つ、スレッドＡの処理後に引き続きスレッドＢの処理が開始される。同様に、図１（ａ）ではスレッドＢのフォーク命令の実行が不可能であるため、そのフォーク命令は無効化されて後続命令の処理が引き続き実行され、その後にスレッドＣの処理が実行されている。この場合のスレッドの実行シーケンスは、Ａ→Ｂ→Ｃであり、図２２（ｂ）で説明した単一のプロセッサによる逐次実行順序と同じであり、プログラムの処理は支障なく遂行できる。

【００２３】図１（ｂ）は、スレッドＡからスレッドＢはフォークできたが、スレッドＢからスレッドＣはフォークできなかった場合の実行シーケンスを示す。この場合、スレッドＡを実行していたプロセッサＰＥ１はスレッドＢをフォークしたので、スレッドＡだけを実行し、プロセッサＰＥ２はスレッドＣをフォークできなかったので、スレッドＢに引き続きスレッドＣを実行する。

【００２４】第１の発明では、フォーク命令の時点で子スレッドの実行を開始できる他プロセッサが存在していないと最早フォークは行えないが、第２及び第３の発明のようなレジスタファイルの更新を考慮したフォーク可能性の判定が不要になるため、制御の簡素化およびハードウェア量の削減が可能である。

【００２５】（２）第２の発明

図１（ｃ）に示すように、１つのプロセッサＰＥ１にスレッドＡを実行させ、プロセッサＰＥ１でスレッドＡを実行しているときにフォーク命令に実行が差しかかると、子スレッドＢの実行を開始することができる他プロセッサが存在するか否かが判定される。図１（ｃ）では、そのようなプロセッサが存在しないため、当該フォーク命令は保留され、プロセッサＰＥ１は後続命令の処理を続行する。そして、スレッドＡのレジスタファイルが更新される前に子スレッドＢの実行を開始することができる他プロセッサＰＥ２が生じたため、フォーク可能と判定され、プロセッサＰＥ２にスレッドＢが生成されて実行が開始される。プロセッサＰＥ２ではスレッドＢ中のフォーク命令に実行が差しかかると、子スレッドＣの実行を開始することができる他プロセッサが存在するか否かが判定される。図１（ｃ）では、そのようなプロセッサが存在しないため、当該フォーク命令は保留され、プロセッサＰＥ２は後続命令の処理を続行する。また、図１（ｃ）では、スレッドＢのレジスタファイルが更新される前に子スレッドＣの実行を開始することができる他プロセッサが生じなかったため、フォークを断念し、プロセッサＰＥ２はスレッドＢの処理を終了すると、引き続きスレッドＣの処理を実行している。

【００２６】また、プロセッサＰＥ１において、スレッドＡのレジスタファイルが更新される前に子スレッドＢの実行を開始することができる他プロセッサが存在しなかった場合はスレッドＡに引き続きスレッドＢを実行し、またスレッドＢの実行中に、スレッドＢのレジスタ

ファイルが更新される前に子スレッドＣの実行を開始することできる他プロセッサが存在しなかった場合はスレッドＢに引き続きスレッドＣも実行する。この際のスレッドの実行シーケンスは図１（ａ）と同じになる。

【００２７】この第２の発明では、フォーク命令の時点でフォークできなくても、レジスタファイルが更新される前に子スレッドの実行を開始できる他プロセッサが生じるとフォークを行うため、第１の発明に比べてフォークされる可能性が高まり、スレッド実行の並列度が向上する。

【００２８】（３）第３の発明

図１（ｄ）に示すように、１つのプロセッサＰＥ１にスレッドＡを実行させ、プロセッサＰＥ１でスレッドＡを実行しているときにフォーク命令に実行が差しかかると、子スレッドＢの実行を開始することができる他プロセッサが存在するか否かが判定される。図１（ｄ）では、そのようなプロセッサが存在しないため、当該フォーク命令は保留され、プロセッサＰＥ１は後続命令の処理を続行する。そして、図１（ｄ）では、スレッドＡのレジスタファイルのうち子スレッドＢに継承すべきレジスタの値が更新される前に子スレッドＢの実行を開始することができる他プロセッサＰＥ２が生じたため、フォーク可能と判定され、プロセッサＰＥ２にスレッドＢが生成されて実行が開始される。プロセッサＰＥ２ではスレッドＢ中のフォーク命令に実行が差しかかると、子スレッドＣの実行を開始することができる他プロセッサが存在するか否かが判定される。図１（ｄ）では、そのようなプロセッサが存在しないため、当該フォーク命令は保留され、プロセッサＰＥ２は後続命令の処理を続行する。また、図１（ｄ）では、スレッドＢのレジスタファイル中の子スレッドＣに継承すべきレジスタの値が更新される前に子スレッドＣの実行を開始することができる他プロセッサが生じなかったため、フォークを断念し、プロセッサＰＥ２はスレッドＢの処理を終了すると、引き続きスレッドＣの処理を実行している。

【００２９】また、プロセッサＰＥ１において、スレッドＡのレジスタファイル中の子スレッドＢに継承すべきレジスタの値が更新される前に子スレッドＢの実行を開始することができる他プロセッサが存在しなかった場合はスレッドＡに引き続きスレッドＢを実行し、またスレッドＢの実行中に、スレッドＢのレジスタファイル中の子スレッドＣに継承すべきレジスタの値が更新される前に子スレッドＣの実行を開始することできる他プロセッサが存在しなかった場合はスレッドＢに引き続きスレッドＣも実行する。この際のスレッドの実行シーケンスは図１（ａ）と同じになる。

【００３０】この第３の発明では、親スレッドのレジスタファイルの更新があっても、その更新が子スレッドに継承すべきレジスタでなければフォークを行うため、第２の発明に比べてフォークされる可能性をより高めることができ、従ってスレッド実行の並列度をより向上することができる。

【００３１】（４）第４の発明

この第４の発明では、従来と同様に子スレッドの開始点の直前にターム命令が置かれる。図１（ｂ）、（ｃ）、（ｄ）において、プロセッサＰＥ１では親スレッドＡから子スレッドＢをフォークしたのでフォークドビットがセットされる。他方、プロセッサＰＥ２では親スレッドＢから子スレッドＣをフォークしなかったのでフォークドビットはセットされない。このため、プロセッサＰＥ１では、子スレッドＢの開始点の直前のターム命令は有効になり、親スレッドＡの実行によりスレッドの処理を終了する。他方、プロセッサＰＥ２では、子スレッドＣの開始点の直前のターム命令は無効化され、これによりスレッドの処理を停止することなく親スレッドＢの実行後に子スレッドＣの命令群の実行が開始される。

【００３２】（５）第５の発明

この第５の発明では、従来と異なり子スレッドの開始点の直前にはターム命令は存在しない。図１（ｂ）、（ｃ）、（ｄ）において、プロセッサＰＥ１はスレッドＡのフォーク命令の時点でフォーク先アドレス（スレッドＢの開始アドレス）をレジスタに保存し、親スレッドＡから子スレッドＢをフォークしたのでフォークドビットがセットされる。他方、プロセッサＰＥ２はスレッドＢのフォーク命令の時点でフォーク先アドレス（スレッドＣの開始アドレス）をレジスタに保存するが、親スレッドＢから子スレッドＣをフォークしなかったのでフォークドビットはセットされない。このため、プロセッサＰＥ１では、プログラムカウンタの値がレジスタに保存されたフォーク先アドレス（スレッドＢの開始アドレス）と一致した時点でスレッドの処理を終了する。他方、プロセッサＰＥ２では、プログラムカウンタの値がレジスタに保存されたフォーク先アドレス（スレッドＢの開始アドレス）と一致してもスレッドの処理は終了せず、子スレッドＣの命令群の処理へと進む。

【００３３】なお、本発明においては、親スレッドから子スレッドへのフォーク時におけるレジスタの値の継承は、フォーク命令時点の親スレッドのレジスタファイルのうち少なくとも子スレッドで必要なレジスタだけを対象とすれば足りる。このための具体的なレジスタ継承機構としては、文献２及び文献３に記載されるようにスレッド生成時に親スレッドのレジスタファイルの内容すべてを子スレッドのレジスタファイルにコピーするものであっても良いし、レジスタ転送量の削減を図るために必要なレジスタの値だけを命令によりレジスタ単位で個別に転送するものであっても良い。

【００３４】

【発明の実施の形態】次に本発明の実施の形態の例について図面を参照して詳細に説明する。

【００３５】

【第１の実施の形態】図２を参照すると、本発明の並列プロセッサシステムの一例は、４スレッド並列実行型プロセッサであり、４個のプロセッサ１－ｉ（ｉ＝０～３）が信号線２－ｉによってスレッド管理部３に接続されると共に、信号線４－ｉによって共有のメモリ５に接続されている。また、プロセッサ１－ｉ相互間は通信バス６で接続されている。この例では、４スレッド並列実行型プロセッサを取り上げたが、８スレッドや１６スレッドの並列実行型プロセッサ等、一般にｎ（≧２）スレッド並列実行型プロセッサに対して本発明は適用可能である。全てのプロセッサ１－ｉ、メモリ５及びスレッド管理部３はクロックに同期して動作する。また、好ましくは、全てのプロセッサ１－ｉはメモリ５及びスレッド管理部３と共に１つの半導体チップ上に集積化される。

【００３６】各プロセッサ１－ｉは、プログラムカウンタ（以下、ＰＣと称す）及びレジスタファイルを独立に有し、ＰＣに従って、メモリ５中のスレッドの命令を同時にフェッチ、解釈、実行する機能を有している。各プロセッサ１－ｉにおけるスレッドの実行は、スレッド管理部３から信号線２－ｉを通じてターゲットＰＣ値を伴うスレッド開始要求７ｃが送信された時点で開始される。スレッドの実行を終了したプロセッサ１－ｉは、スレッド管理部３に対して信号線２－ｉを通じてスレッド終了通知７ｄを送信する。このスレッド終了通知７ｄがスレッド管理部３で受理された時点で、当該プロセッサ１－ｉはフリー状態として管理され、新たなスレッドの実行を当該プロセッサ１－ｉに開始させることができる。

【００３７】各プロセッサ１－ｉは、実行中の親スレッドに存在するフォーク命令によって他のプロセッサ１－ｊ（ｉ≠ｊ）に子スレッドをフォークすることができる。その際、プロセッサ１－ｉは、信号線２－ｉを通じてスレッド管理部３に対し、子スレッドのフォーク先アドレス（開始ＰＣ値）を伴うフォーク要求７ａを送信する。スレッド管理部３は、フォーク要求７ａを受信すると、子スレッドの実行を開始できる他のプロセッサ１－ｊが存在するか否かを調べ、存在すれば当該他のプロセッサ１－ｊに対してフォーク先アドレスを伴うスレッド開始要求７ｃを送信する一方、フォーク要求元のプロセッサ１－ｉに対しては、当該他のプロセッサ１－ｊの番号を指定したフォーク応答７ｂを返却する。この時点で初めてフォークが行われたことになり、フォーク応答７ｂを受信したプロセッサ１－ｉは、フォーク先のプロセッサ１－ｊのレジスタファイルに対して、親スレッドのレジスタファイルの全内容を通信バス６を通じてコピーするか、当該子スレッドで必要なレジスタの値だけをコピーすることにより、レジスタ継承を行う。

【００３８】他方、子スレッドの実行を開始できる他のプロセッサが存在しなかった場合、スレッド管理部３は、今回のフォーク要求７ａを廃棄する。これにより前記フォーク命令は無効化される。

【００３９】図３を参照すると、スレッド管理部３の一例は、スレッド管理シーケンサ１１とプロセッサ状態テーブル１２とから構成される。プロセッサ状態テーブル１２は、プロセッサ１－ｉと１対１に対応するエントリ１３－ｉを有する。個々のエントリ１３－ｉは、対応するプロセッサ１－ｉがビジー状態か、フリー状態かを記録するために使用される。スレッド管理シーケンサ１１は、このプロセッサ状態テーブル１２を用いて各プロセッサ１－ｉにおけるスレッド生成、スレッド終了を管理する。プロセッサ１－ｉからフォーク要求７ａ、スレッド終了通知７ｄを受信した際のスレッド管理シーケンサ１１の処理例を図４及び図５に示す。

【００４０】図４を参照すると、スレッド管理シーケンサ１１は、或るクロックのタイミングで何れかのプロセッサ１－ｉからフォーク要求７ａを受信すると、子スレッドの実行を開始できるプロセッサが存在するか否かをプロセッサ状態テーブル１２を参照して調べる（ステップＳ１）。文献１～３に記載されるように、スレッド管理の簡便化のためにプロセッサ１－ｉから子スレッドをフォークできるプロセッサを、プロセッサ１－ｉの一方の隣接プロセッサ（プロセッサ１－０はプロセッサ１－１、プロセッサ１－１はプロセッサ１－２、プロセッサ１－２はプロセッサ１－３、プロセッサ１－３はプロセッサ１－０）に限定したモデル（このようなモデルを以下、リング型フォークモデルと称す）では、プロセッサ状態テーブル１２におけるプロセッサ１－ｉに隣接するプロセッサ１－ｊに対応するエントリ１３－ｊを参照し、フリー状態であれば子スレッドの実行を開始できるプロセッサが存在すると判定でき、ビジー状態であればそのようなプロセッサは存在しないと判定できる。

【００４１】子スレッドの実行を開始できるプロセッサ１－ｊが存在した場合、スレッド管理シーケンサ１１は、プロセッサ状態テーブル１２における当該プロセッサ１－ｊに対応するエントリ１３－ｊをフリー状態からビジー状態に更新し（ステップＳ２）、フォーク要求７ａに付随するフォーク先アドレスを添えたスレッド開始要求７ｃをフォーク先プロセッサ１－ｊに送信すると共に、要求元のプロセッサ１－ｉに対してフォーク先プロセッサ１－ｊを指定したフォーク応答７ｂを返却する（ステップＳ３）。リング型フォークモデルでは、フォーク先プロセッサは事前に特定されるので、フォーク応答７ｂでフォーク先プロセッサ１－ｊを指定する必要はない。

【００４２】他方、子スレッドの実行を開始できるプロセッサ１－ｊが存在しなかった場合、スレッド管理シーケンサ１１は、当該フォーク要求７ａを破棄する（ステップＳ４）。

【００４３】図５を参照すると、スレッド管理シーケンサ１１は、何れかのプロセッサ１－ｉからスレッド終了

通知７ｄを受信すると、プロセッサ管理テーブル１２における当該プロセッサ１－ｉに対応するエントリ１３－ｉをビジー状態からフリー状態に更新する（ステップＳ１１）。

【００４４】図６を参照すると、各々のプロセッサ１－ｉは、スレッド管理部３から送信されたスレッド開始要求７ｃに付随する開始アドレス値がセットされ、その後に適宜歩進されるＰＣ２１と、ＰＣ２１に従ってメモリ５からスレッドの命令をフェッチする命令フェッチユニット２２と、フェッチされた命令をデコードし、実行する実行ユニット２３と、汎用レジスタ２４－０～２４－ｍの集合であるレジスタファイル２５と、フォーク先プロセッサに対して通信バス６経由でレジスタファイル２５の内容を転送するレジスタ転送ユニット２６と、フォーク命令実行時に実行ユニット２３からスレッド管理部３に送信されたフォーク先アドレスを伴うフォーク要求７ａに対するフォーク応答７ｂによってセットされるフォークドビット２７とを含んで構成され、フォークドビット２７の値は実行ユニット２３に入力されている。

【００４５】実行ユニット２３は、スレッド中のターム命令のデコード時、フォークドビット２７がセットされているか否かを判別し、セットされているときは当該ターム命令を有効に実行することによりスレッドの処理を終了する。この際、スレッド管理部３に対してスレッド終了通知７ｄを送信する。他方、フォークドビット２７がセットされていないときは当該ターム命令を無効化し、ＰＣ２１に従って後続命令の処理を続行する。また、レジスタ転送ユニット２６は、フォークドビット２７がセットされるタイミングでフォーク先プロセッサへのレジスタ転送を開始する。レジスタ転送ユニット２６は、例えば、通信バス６のバス幅によって一度に転送できる数のレジスタ毎に、レジスタファイル２５のレジスタの値とレジスタ番号（レジスタアドレス）とをフォーク先プロセッサのレジスタファイルへ送信し、受信側のレジスタファイル２５では該当するレジスタを書き換える。

【００４６】次に本実施の形態にかかるマルチスレッド実行方法の動作を、スレッドの開始から終了までのプロセッサ１－ｉ及びスレッド管理部３の処理の一例を示す図７のフローチャートを参照して説明する。

【００４７】スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォークドビット２７がリセットされ（ステップＳ２１）、スレッドの命令のフェッチ、デコード、実行が以後継続して実行される（ステップＳ２２）。図７では、スレッド中の命令のうち、ターム命令とフォーク命令について特に注目して処理の概要を例示してある。

【００４８】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ２４でＹＥＳ）、実行ユニット２３はフォーク先アドレスを指定したフォーク要求７ａをスレッド管理部３に送信し、スレッド管理部３は前述したようにして子スレッドの実行を開始できるフリー状態の他プロセッサ１－ｊが存在するか否かを調べる（ステップＳ２５）。フリー状態の他プロセッサ１－ｊが存在した場合、スレッド管理部３は前述したようにプロセッサ１－ｉのフォークドビット２７をセットすると同時に他プロセッサ１－ｊに対してスレッド開始要求７ｃを送出することで子スレッドをフォークする（ステップＳ２６）。この際、プロセッサ１－ｉのレジスタ転送ユニット２６は、レジスタファイル２５の内容を通信バス６経由でプロセッサ１－ｊへ送信し、プロセッサ１－ｊのレジスタファイル２５に書き込む。他方、子スレッドの実行を開始できる他プロセッサが存在しなかった場合、スレッド管理部３はプロセッサ１－ｉから送信されたフォーク要求７ａを廃棄することでフォーク命令を無効化する。プロセッサ１－ｉではフォーク命令の実行後、ＰＣ２１に従って次の命令の実行を継続する（ステップＳ２２）。

【００４９】実行ユニット２３でデコードされた命令がターム命令の場合（ステップＳ２３でＹＥＳ）、実行ユニット２３はフォークドビット２７がセットされていれば（ステップＳ２７でＹＥＳ）、当該ターム命令を実行することによりスレッドの処理を終了する（ステップＳ２８）。しかし、フォークドビット２７がセットされていなければ、当該ターム命令を無効化し、ＰＣ２１に従って次の命令の実行を継続する（ステップＳ２２）。

【００５０】リング型フォークモデルに適用した本実施の形態にかかるマルチスレッド実行方法の実行シーケンスの一例を図８に示す。図８では、プロセッサ＃０からその隣接プロセッサ＃１に、プロセッサ＃１からその隣接プロセッサ＃２に、プロセッサ＃２からその隣接プロセッサ＃３にそれぞれ子スレッドをフォークしている。また、プロセッサ＃３のフォーク点でプロセッサ＃０がフリー状態となっているため、プロセッサ＃３からプロセッサ＃０への子スレッドのフォークも成功している。しかし、このプロセッサ＃０に新たに生成されたスレッドのフォーク点では、隣接プロセッサ＃１がビジー状態であるためフォークは不可能であり、フォークはスキップ（無効化）されている。このため、プロセッサ＃０は当該スレッドのターム命令をスキップ（無効化）し、本来は隣接プロセッサ＃１で行うべき子スレッドを自身で実行している。

【００５１】次に、本実施の形態にかかるマルチスレッド実行方法で実行される並列化プログラムの生成方法について説明する。

【００５２】図９を参照すると、コンパイラ４１は、逐次処理プログラム４２を入力し、制御及びデータフロー解析部４４によって逐次処理プログラム４２の制御フロー及びデータフローを解析して、基本ブロック或いは複

数の基本ブロックを並列化の単位、すなわちスレッドに分割し、次いで並列化コード挿入部４５によって並列化のためのコードを挿入して、複数のスレッドに分割された並列化プログラム４３を生成して出力する。本発明では、並列化プログラム４３中の各スレッドはその生存中に高々１回に限って有効な子スレッドを生成するというフォーク１回制限を並列化プログラム４３において静的に保証している。

【００５３】並列化コードとしては、フォーク命令、ターム命令などがある。本実施の形態では並列化プログラムの生成時に、フォーク点にフォーク命令が挿入され、且つ子スレッドの開始点の直前にターム命令が挿入される。また、前述したように本実施の形態では、フォーク不可能ならばフォーク命令が無効化され、フォーク命令が無効化されたときには対応するターム命令も無効化されるため、コンパイラ４１は、フォーク命令、ターム命令がたとえ無効化されても逐次処理プログラム４２と等価な処理が行える並列化プログラム４３を生成する。一般に、並列化プログラム４３中のフォーク命令及びターム命令を全て取り除いた状態の制御フローが逐次処理プログラム４２の制御フローと等価であれば、逐次処理プログラム４２の動作を保証できる並列化プログラム４３となる。

【００５４】図１０（ａ）に逐次処理プログラム４２の一例を示す。この例の逐次処理プログラム４２は、米国ＭＩＰＳ Ｔｅｃｈｎｏｌｏｇｙ ＩＮＣ．社のＲＩＳＣプロセッサの命令セットを用いて記述されており、レジスタｒ１４に５を加算し（ａｄｄｕ）、レジスタｒ１６の指し示すアドレスにレジスタｒ１４の内容をストアし（ｓｗ）、レジスタｒ１６の指し示すアドレス＋４の内容をレジスタｒ１にロードし（ｌｗ）、ループを実行するよう記述されている。このループでは、＿ｆｕｎｃを開始アドレスとする関数呼び出しを行い（ｊａｌ）、レジスタｒ１の値を－１し（ｓｕｂ）、レジスタｒ１とレジスタｒ０（常に０）を比較し等しくなければ＿ｌｏｏｐへ分岐する（ｂｎｅ）、処理を記述してある。

【００５５】図１０（ｂ）に図１０（ａ）の逐次処理プログラム４２をコンパイラ４１がコンパイルして生成した並列化プログラム４３の一例を示す。この例では、ａｄｄｕ命令の直前に、＿ｔｈ１をフォーク先アドレスとする１つのフォーク命令（ｆｏｒｋ）が挿入され、このフォーク先アドレス＿ｔｈ１の直前にターム命令（ｔｅｒｍ）が挿入されている。また、この子スレッド中に＿ｔｈ２をフォーク先アドレスとする１つのフォーク命令が挿入され、このフォーク先アドレス＿ｔｈ２の直前にターム命令が挿入されている。

【００５６】図１０（ｂ）の並列化プログラム４３からフォーク命令及びターム命令を全て除去すると、図１０（ａ）の逐次処理プログラム４２と同じ制御フローのプログラムとなり、フォーク命令、ターム命令が無効化されても逐次処理プログラム４２で遂行される処理が保証される並列化プログラムとなっているのが分かる。

【００５７】なお、図１０（ｂ）の並列化プログラム４３には、親スレッドのレジスタファイル中のレジスタのうち、子スレッドに継承すべきレジスタの情報が含まれていない為、子スレッドのフォーク時には親スレッドのレジスタファイルの内容を全て転送することになる。勿論、後述する第３の実施の形態と同様に子スレッドに継承すべきレジスタをコンパイル時に解析して並列化プログラム４３に継承レジスタの情報を記述しておき、子スレッドのフォーク時に親スレッドのレジスタファイルの内容のうち子スレッドで必要なレジスタだけを転送するように構成することも可能である。

【００５８】

【第２の実施の形態】本実施の形態は、並列化プログラム中のターム命令を不要にした点で第１の実施の形態と相違する。以下、第１の実施の形態との相違点を中心に本実施の形態を説明する。

【００５９】図１１を参照すると、本実施の形態における並列プロセッサシステムの各々のプロセッサ１－ｉは、図６に示した構成に加えて、実行ユニット２３からスレッド管理部３に送信されるスレッド要求７ａに付随するフォーク先アドレスを保存するレジスタ２８と、ＰＣ２１の値がレジスタ２８に保存されたフォーク先アドレスと一致するか否かを判定する一致回路２９と、フォークドビット２７及び一致回路２９の出力の論理積出力を実行ユニット２３に出力するアンドゲート３０とを含んで構成されている。

【００６０】実行ユニット２３は、フォークドビット２７がセットされている状態においてＰＣ２１の値がレジスタ２８に保存されたフォーク先アドレスと一致することによりアンドゲート３０の出力が論理“１”になると、スレッドの処理を終了し、スレッド管理部３に対してスレッド終了通知７ｄを送信する。ＰＣ２１の値がレジスタ２８に保存されたフォーク先アドレスと一致しても、フォークドビット２７がセットされていなければ、アンドゲート３０の出力は論理“１”にならないため、実行ユニット２３はＰＣ２１に従って命令の実行を継続する。

【００６１】本実施の形態におけるスレッドの開始から終了までのプロセッサ１－ｉ及びスレッド管理部３の処理の一例を図１２に示す。図７との相違点はステップＳ３１、Ｓ３２である。

【００６２】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ２４でＹＥＳ）、実行ユニット２３はフォーク命令で指定されたフォーク先アドレスをレジスタ２８に保存し（ステップＳ３２）、このレジスタ２８の出力を伴ってフォーク要求７ａがスレッド管理部３に送信される。スレッド管理部３は第１の実施の形態と同様に子スレッドの実行を開始できるフリ

一状態の他プロセッサ１－ｊが存在すれば（ステップＳ２５でＹＥＳ）、プロセッサ１－ｉのフォークドビット２７をセットすると同時に他プロセッサ１－ｊに対してスレッド開始要求７ｃを送出することで子スレッドをフォークする（ステップＳ２６）。他方、子スレッドの実行を開始できる他プロセッサが存在しなかった場合、スレッド管理部３はプロセッサ１－ｉから送信されたフォーク要求７ａを廃棄することでフォーク命令を無効化する。

【００６３】プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２８に保存されたフォーク先アドレスに一致すると（ステップＳ３１でＹＥＳ）、フォークドビット２７がセットされていれば（ステップＳ２７でＹＥＳ）、アンドゲート３０の出力が論理“１”となり、実行ユニット２３に割り込みがかかり、当該プロセッサ１－ｉはスレッドの処理を終了する（ステップＳ２８）。しかし、フォークドビット２７がセットされていなければ、ＰＣ２１に従って命令の実行、つまり子スレッド命令群を継続して実行することになる（ステップＳ２２）。

【００６４】上述したように本実施の形態にかかるマルチスレッド実行方法では、並列化プログラム中のターム命令が不要になるため、図９のコンパイラ４１における並列化コード挿入部４５は、子スレッドの開始点の直前にターム命令は挿入しない。図１３に、図１０（ａ）の逐次処理プログラム４２を本実施の形態にかかるマルチスレッド実行方法向けに生成した並列化プログラム４３の一例を示す。図１０（ｂ）の並列化プログラムの子スレッドの開始点（－ｔｈ１、－ｔｈ２）の直前に挿入されていたターム命令は、図１３の並列化プログラムでは省略されている。

【００６５】

【第３の実施の形態】第１及び第２の実施の形態では、親スレッドのフォーク点でフォーク可能でなければフォークを即断念したが、本実施の形態では、親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他プロセッサが生じるとフォークを行う。以下、第２の実施の形態との相違点を中心に本実施の形態を説明する。

【００６６】図１４を参照すると、本実施の形態における並列プロセッサシステムの各々のプロセッサ１－ｉは、図１１に示した構成に加えて、フォーク有効ビット３１を含んで構成されている。フォーク有効ビット３１は、実行ユニット２３がフォーク命令を実行したときに出力するフォーク信号３７でセットされ、スレッド管理部３から受信されるフォーク応答７ｂ及び実行ユニット２３が親スレッドのレジスタファイル２５中の何れかのレジスタを更新したときに出力するレジスタ更新信号３３によってリセットされる。フォーク有効ビット３１の出力がスレッド管理部３に対するフォーク要求７ａとなり、フォーク有効ビット３１がセットされている間、フォーク要求７ａが送出し続けられる。

【００６７】前述の図４を参照すると、スレッド管理部３のスレッド管理シーケンサ１１は、或るクロックのタイミングでプロセッサ１－ｉからフォーク要求７ａを受信した際、子スレッドの実行を開始できるプロセッサが存在しないとき（ステップＳ１でＮＯ）、当該フォーク要求７ａは破棄したが、前述したように、プロセッサ１－ｉはフォーク有効ビット３１がセットされている間、フォーク要求７ａを送出し続けているので、次回のクロックのタイミングでスレッド管理部３がプロセッサ１－ｉからフォーク要求７ａを再び受信することになり、図４の処理が繰り返される。即ち、フォーク点でフォーク不可能な場合、フォーク命令は保留にされ、フォーク可能となった時点で実行されることになる。

【００６８】次に本実施の形態にかかるマルチスレッド実行方法の動作を、スレッドの開始から終了までのプロセッサ１－ｉ及びスレッド管理部３の処理の概要を示す図１５のフローチャートを参照して説明する。

【００６９】スレッド管理部３からのスレッド開始要求７ｃに基づき、プロセッサ１－ｉで１つのスレッドの実行が開始される際、当該プロセッサ１－ｉのフォーク有効ビット３１及びフォークドビット２７がリセットされ（ステップＳ４１）、スレッドの命令のフェッチ、デコード、実行が以後継続して実行される（ステップＳ４２）。図１５では、スレッドの終了に関する処理とフォーク命令の処理について特にその概要を例示してある。

【００７０】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ４４でＹＥＳ）、実行ユニット２３はフォーク先アドレスをレジスタ２８に保存すると共にフォーク信号３７を出力してフォーク有効ビット３１をセットする（ステップＳ４５）。また、実行ユニット２３はレジスタファイル２５中の何れかのレジスタを更新すると（ステップＳ４６でＹＥＳ）、レジスタ更新信号３３を出力してフォーク有効ビット３１をリセットする（ステップＳ４７）。従って、プロセッサ１－ｉからは、フォーク命令実行時点からレジスタファイル２５が最初に更新される迄の期間中、フォーク要求７ａがスレッド管理部３に送出し続けられる。

【００７１】スレッド管理部３は、或るクロックのタイミングでプロセッサ１－ｉからフォーク要求７ａを受信すると、図４に示したように、子スレッドの実行を開始できるフリー状態の他プロセッサ１－ｊが存在するか否かを調べ（ステップＳ１）、フリー状態の他プロセッサ１－ｊが存在した場合にはプロセッサ状態テーブル１２を前述したように更新し（ステップＳ２）、プロセッサ１－ｉにフォーク応答７ｂを送信すると同時に他プロセッサ１－ｊに対してスレッド開始要求７ｃを送出することで子スレッドをフォークする（ステップＳ３）。プロセッサ１－ｉに出されたフォーク応答７ｂによって、フ

ォークドビット２７がセットされ、フォーク有効ビット３１はリセットされる。他方、子スレッドの実行を開始できる他プロセッサが存在しなかった場合（ステップＳ１でＮＯ）、スレッド管理部３は今回のプロセッサ１－ｉから送信されたフォーク要求７ａを破棄して図４の処理を終了するが、前述したようにプロセッサ１－ｉからはフォーク要求７ａが送出し続けられている。図１５のステップＳ５１～Ｓ５３は以上のような処理を別の観点でフローチャート化したものであり、プロセッサ１－ｉのフォーク有効ビット３１がセットされ且つフリー状態のプロセッサ１－ｊが存在していれば、子スレッドのフォークを行い、フォーク要求元プロセッサ１－ｉのフォークドビット２７をセットし且つフォーク有効ビット３１をリセットすることを示している。なお、プロセッサ１－ｉでは、レジスタ転送ユニット２６によるレジスタファイル２５の転送中、実行ユニット２３からレジスタファイル２５への書き込みは待たされる。

【００７２】なお、第２の実施の形態と同様に、プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２８に保存されたフォーク先アドレスに一致すると（ステップＳ４３でＹＥＳ）、フォークドビット２７がセットされていれば（ステップＳ４８でＹＥＳ）、アンドゲート３０の出力が論理“１”となり、実行ユニット２３に割り込みがかかって当該プロセッサ１－ｉはスレッドの処理を終了する（ステップＳ４９）。

【００７３】リング型フォークモデルに適用した本実施の形態にかかるマルチスレッド実行方法の実行シーケンスの一例を図１６に示す。図８とほぼ同じ状況を想定しており、プロセッサ＃３からプロセッサ＃０にフォークされたスレッドのフォーク点Ａでは、隣接プロセッサ＃１はビジー状態である。このような状況の場合、第１及び第２の実施の形態ではフォークを即断念したが、本実施の形態ではフォーク点Ａでフォーク先アドレスをレジスタ２８に保存し、フォークを保留状態とする。フォーク点Ａから下に延びる矢印は、プロセッサ＃０においてレジスタファイル２５が全く更新されていない期間を示す。図１６では、この期間内にプロセッサ＃１がフリー状態になったため、保留されたフォークが実行され、プロセッサ＃１に子スレッドが生成されている。また、プロセッサ＃０は子スレッドを生成したため、フォーク先アドレスに到達するとスレッドの処理を終了している。

【００７４】なお、図１４に示される各プロセッサ１－ｉの実行ユニット２３が、ターム命令のデコード時、アンドゲート３０の出力が論理“１”か否かを判別し、論理“１”ならば当該ターム命令を実行することによりスレッドの処理を終了し、論理“１”でなければ当該ターム命令を無効化し、ＰＣ２１に従って後続命令の処理を続行するように構成すれば、第１の実施の形態と同様に、子スレッドの開始点の直前にターム命令が挿入された並列化プログラムを支障なく実施する別の実施の形態が得られる。

【００７５】【第４の実施の形態】第３の実施の形態では、親スレッドのフォーク点でフォーク可能でなければフォークを一旦保留にし、親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他プロセッサが生じなかった場合に当該フォークを断念したが、本実施の形態では、親スレッドのレジスタファイルが更新されても、その更新が子スレッドに継承すべきレジスタでなければフォークを行う。以下、第３の実施の形態との相違点を中心に本実施の形態を説明する。

【００７６】図１７を参照すると、本実施の形態における並列プロセッサシステムの各々のプロセッサ１－ｉは、図１４に示した構成に加えて、レジスタファイル２５の各レジスタ２４－ｋ（ｋ＝０～ｍ）に１対１に対応し、対応するレジスタ２４－ｋが子スレッドへ継承すべきレジスタであるときに限りセットされるクリエイトビット３２－ｋと、各レジスタ２４－ｋに１対１に対応し、対応するレジスタ２４－ｋのクリエイトビット３２－ｋの出力と実行ユニット２３がレジスタ２４－ｋを更新したときに出力するレジスタ更新信号３３－ｋとを入力とするアンドゲート３４－ｋと、アンドゲート３３－ｋの出力の論理和信号であるフォーク無効信号３５を出力するオアゲート３６とを含んで構成されている。そして、図１４のレジスタ更新信号３３に代えて、フォーク無効信号３５がフォーク有効ビット３１にリセット信号として出力されている。また、各クリエイトビット３２－ｋの値がレジスタ転送ユニット２６に出力されており、レジスタ転送ユニット２６はレジスタファイル２５のレジスタ２４－ｋのうち、対応するクリエイトビット３２－ｋがセットされているレジスタのみをフォーク先プロセッサのレジスタファイルに転送するように構成されている。

【００７７】本実施の形態におけるスレッドの開始から終了までのプロセッサ１－ｉ及びスレッド管理部３の処理の概要を図１８に示す。図１５との相違点はステップＳ６１、Ｓ６２である。

【００７８】実行ユニット２３でデコードされた命令がフォーク命令の場合（ステップＳ４４でＹＥＳ）、実行ユニット２３はフォーク先アドレスをレジスタ２８に保存すると共にフォーク信号３７を出力してフォーク有効ビット３１をセットし、且つ全てのクリエイトビット３２－ｋのセットアップを行う（ステップＳ６１）。即ち、レジスタファイル２５のレジスタ２４－ｋのうち、子スレッドに継承すべきレジスタに対応するクリエイトビット３２－ｋはセットし、継承する必要のないレジスタに対応するクリエイトビット３２－ｋはリセットされたままにする。また、実行ユニット２３はレジスタファイル２５中の何れかのレジスタ２４－ｋを更新すると、その更新したレジスタ２４－ｋに対応するレジスタ更新

信号３３－ｋを論理“１”とする。これにより、若し更新されたレジスタ２４－ｋが子スレッドへ継承すべきレジスタであった場合、そのレジスタ２４－ｋに対応するクリエイトビット３２－ｋはセットされているため、そのレジスタ２４－ｋに対応するアンドゲート３４－ｋの出力が論理“１”となり、オアゲート３６からフォーク無効信号３５が出力されてフォーク有効ビット３１がリセットされる（ステップＳ６２、Ｓ４７）。つまり、プロセッサ１－ｉからは、フォーク命令実行時点からレジスタファイル２５中の子スレッドへの継承レジスタの何れかが最初に更新される迄の期間中、フォーク要求７ａがスレッド管理部３に送出し続けられる。

【００７９】スレッド管理部３は、第３の実施の形態と同様の処理を行う。これによって、プロセッサ１－ｉのレジスタファイル２４のレジスタ２４－ｋのうち、子スレッドへ継承すべきレジスタが更新される前に子スレッドの実行を開始できる他プロセッサが生じると、プロセッサ１－ｉにフォーク応答７ｂを送信すると同時に他プロセッサ１－ｊに対してスレッド開始要求７ｃを送出することで子スレッドをフォークする（ステップＳ５１～５３）。フォーク応答７ｂを受信したプロセッサ１－ｉのレジスタ転送ユニット２６は、レジスタファイル２５のレジスタ２４－ｋのうち、対応するクリエイトビット３２－ｋがセットされているレジスタのみをフォーク先プロセッサのレジスタファイルへ転送する。

【００８０】なお、第３の実施の形態と同様に、プロセッサ１－ｉで命令の実行が進み、ＰＣ２１の値がレジスタ２８に保存されたフォーク先アドレスに一致すると（ステップＳ４３でＹＥＳ）、フォークドビット２７がセットされていれば（ステップＳ４８でＹＥＳ）、アンドゲート３０の出力が論理“１”となり、実行ユニット２３に割り込みがかかって当該プロセッサ１－ｉはスレッドの処理を終了する（ステップＳ４９）。

【００８１】リング型フォークモデルに適用した本実施の形態にかかるマルチスレッド実行方法の実行シーケンス例は図１６と同じようになる。但し、フォーク点Ａから下に延びる矢印は、本実施の形態ではプロセッサ＃０においてレジスタファイル２５のうち子スレッドに継承すべきレジスタが全く更新されていない期間となり、その分だけフォーク可能期間が延長される。

【００８２】本実施の形態では、親スレッドのフォーク点で子スレッドへ継承すべきレジスタが判明している必要がある。このため、図９に示したコンパイラ４１における制御及びデータフロー解析部４４では、フォークする子スレッド毎に、親スレッドから子スレッドへ継承すべきレジスタを調査し、並列化コード挿入部４５ではその調査結果に基づいて、子スレッドへ継承すべきレジスタを指定する記述を並列化プログラム４３に挿入する。

【００８３】図１９に、図１０（ａ）の逐次処理プログラム４２を本実施の形態にかかるマルチスレッド実行方法向けに生成した並列化プログラム４３の一例を示す。１行目のフォーク命令中の「ｒ１，ｒ１６，ｓｐ」、５行目のフォーク命令中の「ｒ１，ｓｐ」がそれぞれ子スレッドへ継承すべきレジスタの指定記述である。各プログラム１－ｉの実行ユニット２３はフォーク命令のデコード時、このような継承レジスタの指定を解釈し、指定されたレジスタ２４－ｋに対応するクリエイトビット３２－ｋのみをセットする。

【００８４】図２０に、図１０（ａ）の逐次処理プログラム４２を本実施の形態にかかるマルチスレッド実行方法向けに生成した並列化プログラム４３の別の例を示す。図１９のようにフォーク命令で継承レジスタを指定する方法では、フォーク命令の命令幅が大きくなるが、本例では、クリエイト（ｃｒｅａｔｅ）命令という特殊命令を定義し、このクリエイト命令で子スレッドへ継承すべきレジスタを指定するため、フォーク命令の命令幅の増大が抑えられる。但し、クリエイト命令という特殊命令が追加されるため命令数は増加する。このため、クリエイト命令が存在しない場合には全レジスタ継承（或いはシステムで事前に設定された所定のレジスタ群の継承）としておき、クリエイト命令が存在すればそれで指定されたレジスタだけを継承するものとして扱う。従って、図２０の１行目のフォーク命令にはその直前にクリエイト命令が存在しないので、全レジスタ継承と認識され、６行目のフォーク命令にはその直前にレジスタｒ１、ｓｐを指定するクリエイト命令が存在するので、レジスタｒ１、ｓｐだけが継承対象となる。

【００８５】図２１に、図１０（ａ）の逐次処理プログラム４２を本実施の形態にかかるマルチスレッド実行方法向けに生成した並列化プログラム４３の更に別の例を示す。図２０ではフォーク命令の直前にクリエイト命令を挿入したが、本例ではフォーク命令の直後にクリエイト命令を挿入している。フォーク命令の直前にクリエイト命令を挿入すると、逐次動作で実行すべき命令が１命令増えるが、フォーク命令の直後にクリエイト命令を挿入するとフォーク命令をなるべく早く実行することができることによって並列に動作する部分を増やすことができる。

【００８６】本実施の形態におけるレジスタ転送ユニット２６は、クリエイトビット３２－ｋを参照することにより、親スレッドのレジスタファイル２５のうち子スレッドに継承すべきレジスタだけをフォーク先プロセッサのレジスタファイルに転送するようにしたが、別の実施例として、レジスタファイル２５の先頭のレジスタから順に所定の順番でレジスタの転送を行うシーケンスを開始し、クリエイトビット３２－ｋがセットされているレジスタの全ての転送が完了した時点で転送シーケンスを停止するようにしても良い。この方法では、子スレッドに継承する必要のないレジスタも転送される場合があるが、転送シーケンスが簡素化される利点がある。勿論、

別の実施例として、クリエイトビット３２－ｋを一切参照せずに常に全レジスタを転送するようにレジスタ転送ユニット２６が構成されていても良い。更に、子スレッドに継承すべきレジスタでも、フォーク先プロセッサの当該レジスタの値がフォーク時点で既に親スレッド側と同じ値になっている場合にはあえて転送する必要がない点に着目して、子スレッドに継承すべきレジスタのうち、親スレッド側と異なる値になっているレジスタを検出し、この検出したレジスタだけをレジスタ転送ユニット２６からフォーク先プロセッサに転送するようにしても良い。

【００８７】なお、図１７に示される各プロセッサ１－ｉの実行ユニット２３が、ターム命令のデコード時、アンドゲート３０の出力が論理“１”か否かを判別し、論理“１”ならば当該ターム命令を実行することによりスレッドの処理を終了し、論理“１”でなければ当該ターム命令を無効化し、ＰＣ２１に従って後続命令の処理を続行するように構成すれば、第１の実施の形態と同様に、子スレッドの開始点の直前にターム命令が挿入された並列化プログラムを支障なく実施する別の実施の形態が得られる。

【００８８】以上、本発明を幾つかの実施の形態を挙げて説明したが、本発明は以上の実施の形態にのみ限定されず、その他各種の付加変更が可能である。例えば、前記各実施の形態では、複数のプロセッサに共通にスレッド管理部３を設ける集中スレッド管理型の並列プロセッサシステムに本発明を適用したが、文献１等に記載されるように各プロセッサ毎にスレッド管理部を設ける分散スレッド管理型の並列プロセッサシステムにも本発明は適用可能である。また、プロセッサ相互間を通信バス６によって接続したが、リング型フォークモデルにあっては隣接するプロセッサ間どうしをリング上に通信線で接続する形態の並列プロセッサシステムに対しても本発明は適用可能である。

【００８９】

【発明の効果】以上説明したように本発明によれば、多数のレジスタファイルを持つことによるハードウェア量の増加、ＯＳのプロセス切り替え時におけるオーバヘッドの増大を防止しつつ、親スレッドのフォーク命令時点で子スレッドを生成できる空きのプロセッサが存在しない場合でも処理の中断無しにプログラムの処理を支障なく遂行することができる効果がある。

【００９０】また第２の発明によれば、フォーク命令の時点でフォークできなくても、レジスタファイルが更新される前に子スレッドの実行を開始できる他プロセッサが生じるとフォークが可能になるため、第１の発明に比べてフォークできる確率が高まり、スレッド実行の並列度を向上することができる。

【００９１】また第３の発明によれば、親スレッドのレジスタファイルの更新があっても、その更新が子スレッドに継承すべきレジスタでなければフォークを行うため、第２の発明に比べてフォークできる確率を高めることができ、スレッド実行の並列度をより一層向上することができる。

【００９２】また第４の発明によれば、従来と同様に子スレッドの開始点の直前にターム命令が置かれたプログラムを支障なく実行することが可能となる。

【００９３】また第５の発明によれば、子スレッドの開始点の直前にターム命令を置く必要がなくなり、ターム命令の削減によってプログラムサイズをコンパクト化でき、命令メモリに必要な容量の削減、命令フェッチ数の削減による処理性能の向上が可能となる。

【図面の簡単な説明】

【図１】本発明の作用の説明図である。

【図２】本発明の並列プロセッサシステムの一例を示すブロック図である。

【図３】本発明の並列プロセッサシステムにおけるスレッド管理部の構成例を示すブロック図である。

【図４】本発明の並列プロセッサシステムにおけるスレッド管理部のスレッド管理シーケンサがプロセッサからフォーク要求を受信した際の処理例を示すフローチャートである。

【図５】本発明の並列プロセッサシステムにおけるスレッド管理部のスレッド管理シーケンサがプロセッサからスレッド終了通知を受信した際の処理例を示すフローチャートである。

【図６】本発明の並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図７】本発明の並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサ及びスレッド管理部の処理の一例を示すフローチャートである。

【図８】リング型フォークモデルに適用した本発明のマルチスレッド実行方法の実行シーケンスの一例を示す図である。

【図９】本発明のマルチスレッド実行方法向けの並列化プログラムを生成するコンパイラの構成例を示すブロック図である。

【図１０】逐次処理プログラムとそれから生成された並列化プログラムの一例を示す図である。

【図１１】本発明の並列プロセッサシステムにおけるプロセッサの別の構成例を示すブロック図である。

【図１２】本発明の並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサ及びスレッド管理部の処理の他の例を示すフローチャートである。

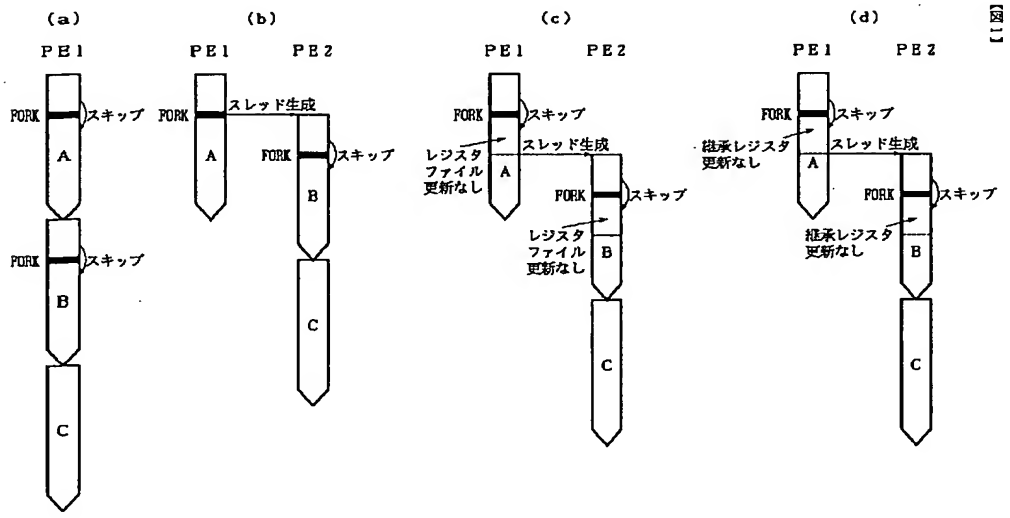
【図１３】逐次処理プログラムから生成された並列化プログラムの他の例を示す図である。

【図１４】本発明の並列プロセッサシステムにおけるプロセッサの更に別の構成例を示すブロック図である。

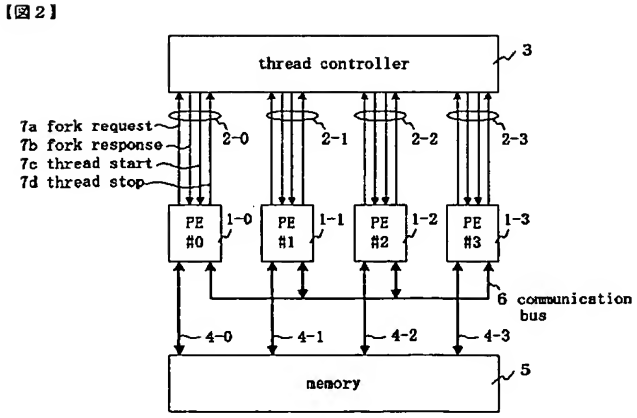
【図１５】本発明の並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサ及びスレッド管

理部の処理の更に別の例を示すフローチャートである。
【図１６】リング型フォークモデルに適用した本発明のマルチスレッド実行方法の実行シーケンスの別の例を示す図である。
【図１７】本発明の並列プロセッサシステムにおけるプロセッサのまた更に別の構成例を示すブロック図である。
【図１８】本発明の並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサ及びスレッド管理部の処理のまた更に別の例を示すフローチャートである。
【図１９】逐次処理プログラムから生成された並列化プログラムの更に別の例を示す図である。
【図２０】逐次処理プログラムから生成された並列化プログラムのまた更に別の例を示す図である。
【図２１】逐次処理プログラムから生成された並列化プログラムの他の例を示す図である。
【図２２】従来のマルチスレッド実行方法の処理の概要を示す図である。
【符号の説明】
１－０～１－３…プロセッサ
２－０～２－３…信号線
３…スレッド管理部
４－０～４－３…信号線
５…メモリ
６…通信バス

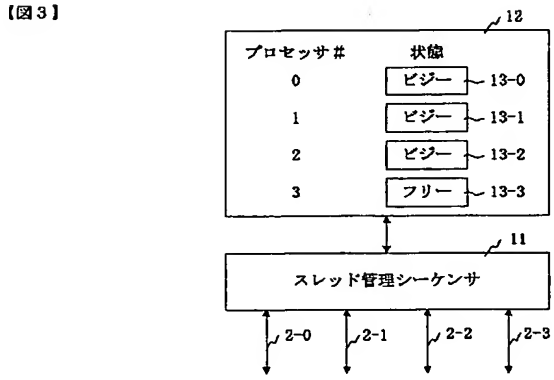
【図１】



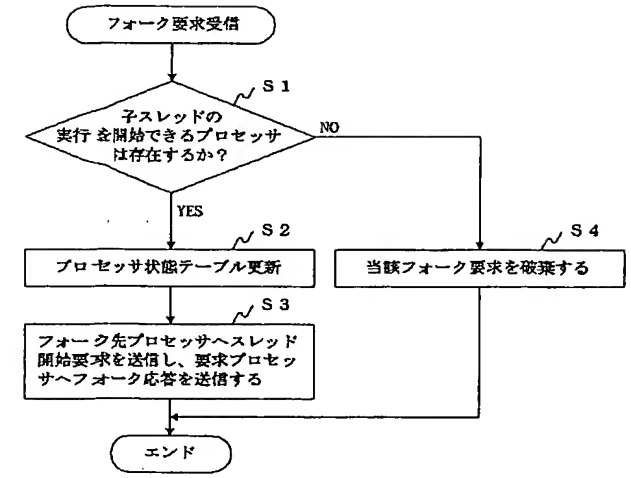
【図２】



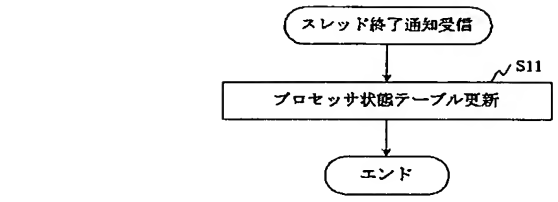
【図３】



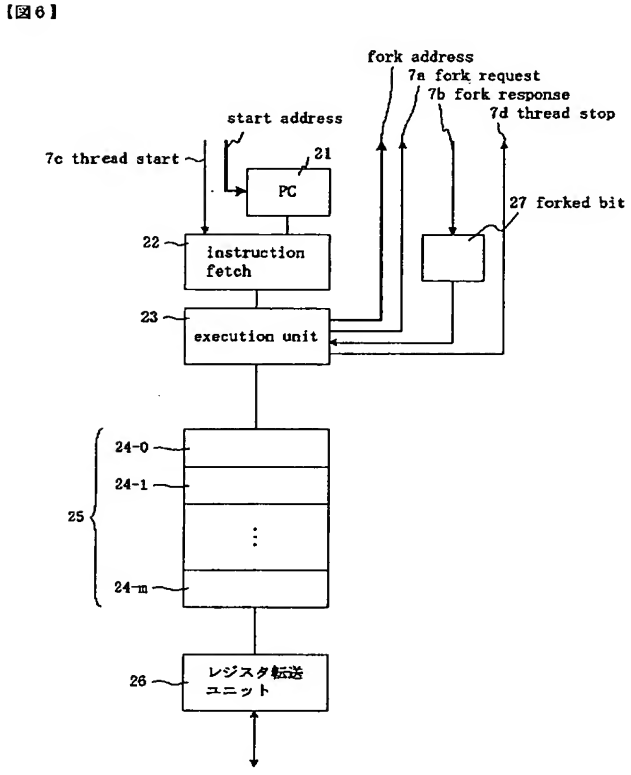
【図４】


【図４】
フォーク要求受信
S１
子スレッドの
実行を開始できるプロセッサ
は存在するか？
NO
YES
S２
プロセッサ状態テーブル更新
S４
当該フォーク要求を破棄する
S３
フォーク先プロセッサへスレッド
開始要求を送信し、要求プロセッ
サへフォーク応答を送信する
エンド

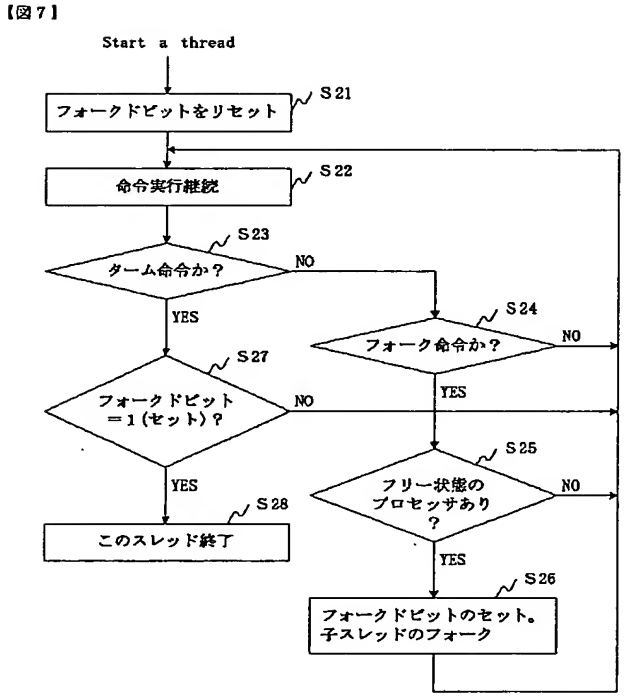
【図５】


【図５】
スレッド終了通知受信
S11
プロセッサ状態テーブル更新
エンド

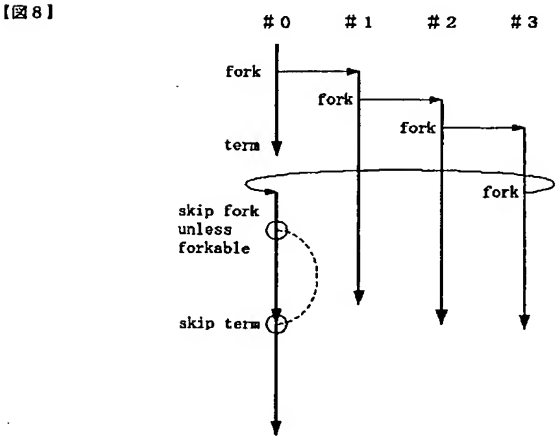
【図６】


【図６】
fork address
7a fork request
7b fork response
7d thread stop
start address
7c thread start
21
PC
27 forked bit
22
instruction
fetch
23
execution unit
24-0
24-1
25
24-m
26
レジスタ転送
ユニット

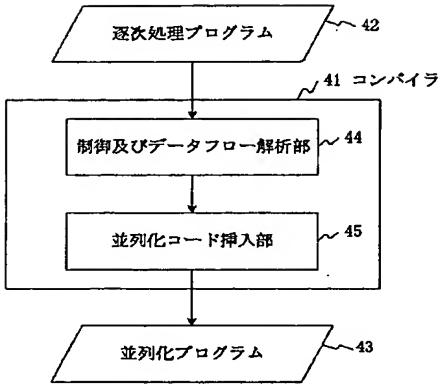
【図７】


【図７】
Start a thread
フォークドビットをリセット
S21
命令実行継続
S22
S23
ターム命令か？
NO
YES
S24
フォーク命令か？
NO
YES
S27
フォークドビット
＝１(セット)？
NO
YES
S25
フリー状態の
プロセッサあり
？
NO
YES
S28
このスレッド終了
S26
フォークドビットのセット。
子スレッドのフォーク

【図８】


【図８】
＃０
＃１
＃２
＃３
fork
fork
fork
term
fork
skip fork
unless
forkable
skip term

【図９】

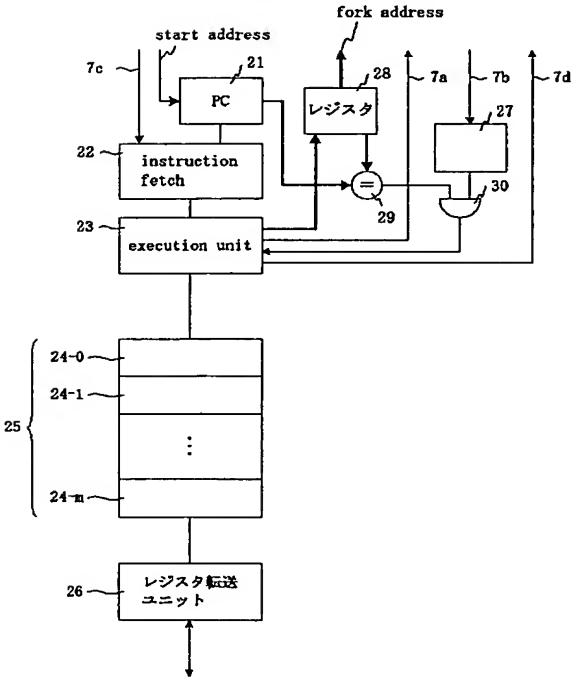


【図１０】

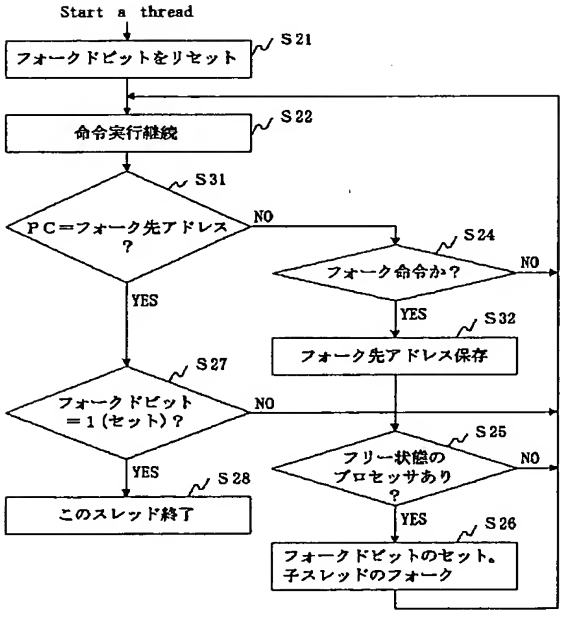
```
(a)             addu    r14, r14, 5
                sw      r14, (r16)
                lw      r1, 4(r16)
        _loop:  jal     _func
                sub     r1, r1, 1
                bne     r1, r0, _loop

                fork    _th1
                addu    r14, r14, 5
                sw      r14, (r16)
                term
(b)     _th1:   lw      r1, 4(r16)
        _loop:  fork    _th2
                jal     _func
                term
        _th2:   sub     r1, r1, 1
                bne     r1, r0, _loop
```

【図１１】



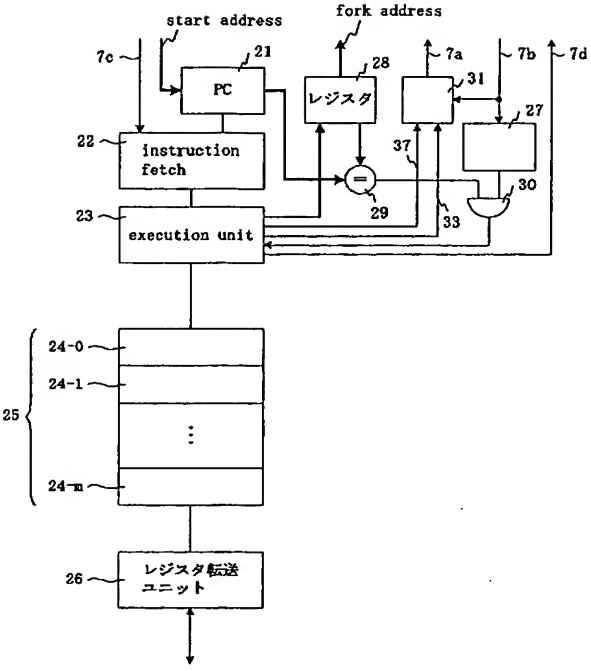
【図１２】



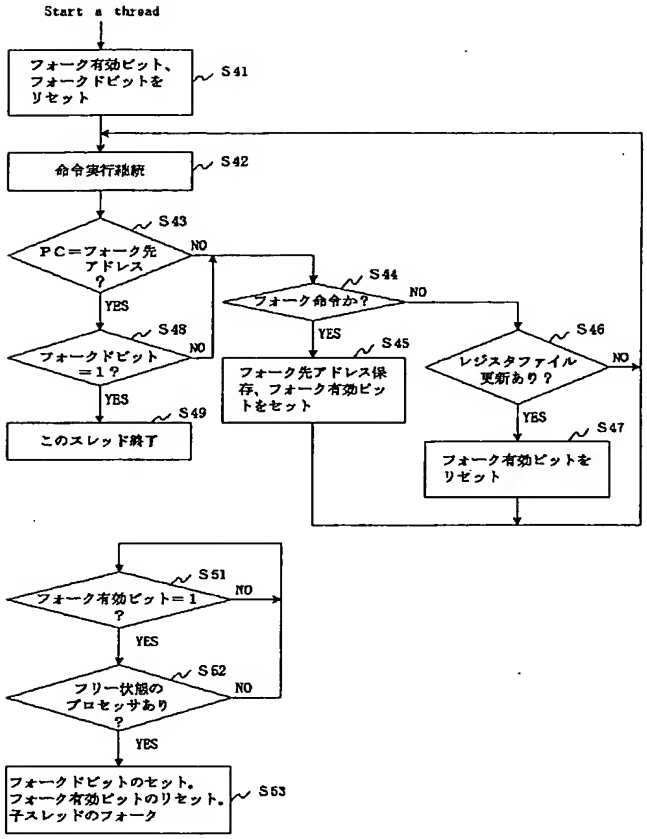
【図１３】

```
                fork    _th1
                addu    r14, r14, 5
                sw      r14, (r16)
        _th1:   lw      r1, 4(r16)
        _loop:  fork    _th2
                jal     _func
        _th2:   sub     r1, r1, 1
                bne     r1, r0, _loop
```

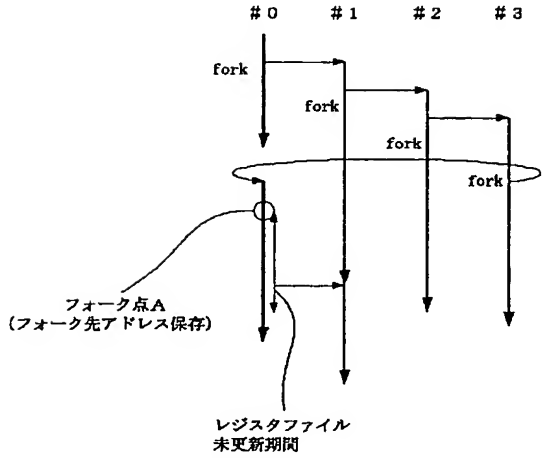
【図１４】



【図１５】



【図１６】



【図１９】

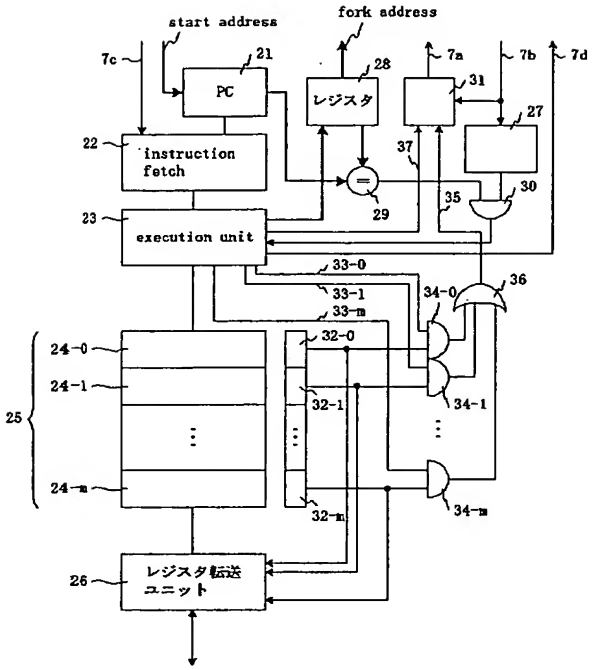
```
        fork    _th1, r1, r16, sp
        addu    r14, r14, 5
        sw      r14, (r16)
_th1:   lw      r1, 4(r16)
_loop:  fork    _th2, r1, sp
        jal     _func
_th2:   sub     r1, r1, 1
        bne     r1, r0, _loop
```

【図２０】

```
        fork    _th1,
        addu    r14, r14, 5
        sw      r14, (r16)
_th1:   lw      r1, 4(r16)
_loop:  create  r1, sp
        fork    _th2
        jal     _func
_th2:   sub     r1, r1, 1
        bne     r1, r0, _loop
```

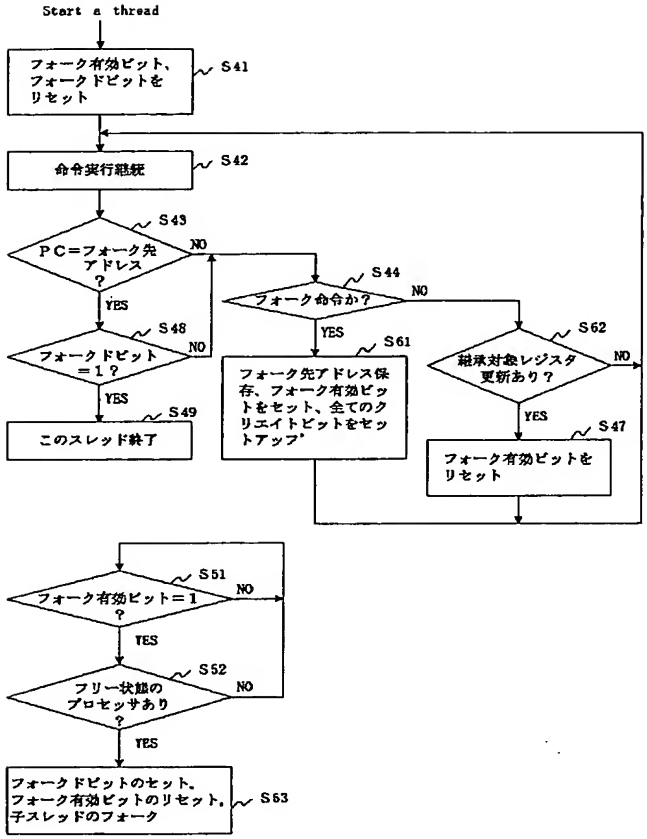
【図１７】

【図17】



【図１８】

【図18】



【図２１】

【図21】

```
        fork    _th1,
        addu    r14,r14,5
        sw      r14,(r16)
_th1:   lw      r1,4(r16)
_loop:  fork    _th2
        create  r1,sp
        jal     _func
_th2:   sub     r1,r1,1
        bne     r1,r0,_loop
```

【図２２】